

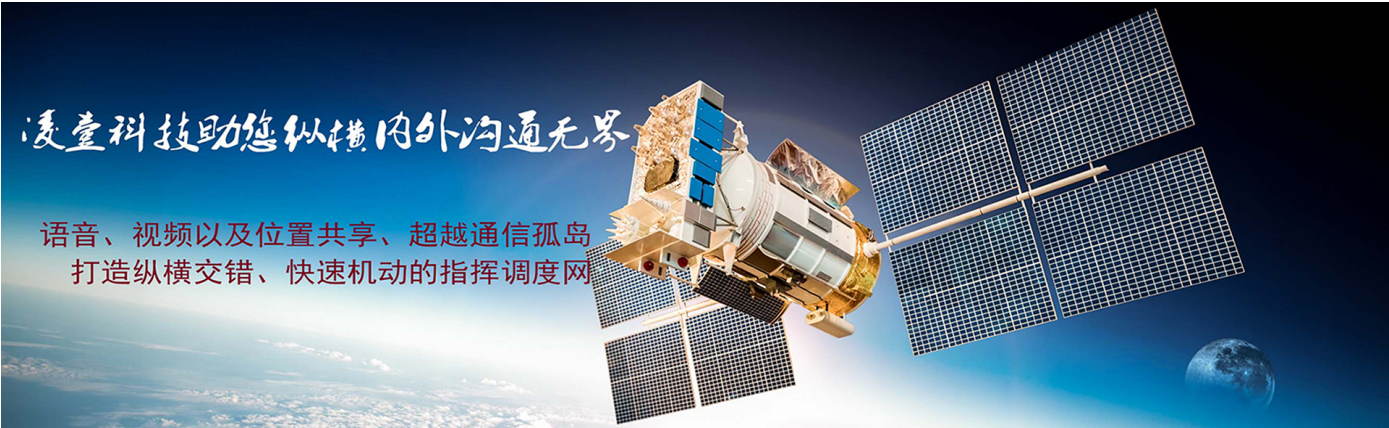
- [T01 Android 端音视频 SDK API 文档](#)
 - [版本](#)
 - [音视频 SDK](#)
 - [USBCamera SDK](#)
 - [注意](#)
 - [快速集成](#)
 - [RegisterEngine](#)
 - [PttEngine](#)
 - [CallEngine](#)
 - [MeetingEngine](#)
 - [MessageEngine](#)
 - [ContactsEngine](#)
 - [LocationEngine](#)
 - [SetEngine](#)
 - [USBCamera SDK 使用](#)

T01 Android 端音视频 SDK API 文档

[公司官网](#)

[GitHub SDKDEMO 实时更新动态](#)





版本

音视频 SDK

- 支持语音通话、语音会议、视频通话
- 320P、480P、720P、1080P 分辨率
- 支持 txt、图片、短视频、文件发送
- 支持 PTT 对讲功能
- 支持视频硬编码

版本	功能	负责人
...	...	
1.0.8	增加 YUV > MediaCodec 硬编码	刘扬, 阳坤
1.0.9	1.更新默认分辨率内部代码 2.内部代码优化	刘扬, 阳坤
1.0.1.0	1. 增加同步查询是否在线接口； 2. 获取组织接口支持传入 unitID 查询; 3.增加查询当前组织ID接口	刘扬, 阳坤
		刘

1.0.1.1	1. ContactsEngine 增加按时间轮训请求在线用户;2. 优化对讲组人数在上万的情况下获取数据慢的性能问题;	刘 扬, 阳坤
1.0.1.2	1. 增加视频多路呼叫接口。详细信息请看 CallEngine 接口; 2. 修改 PttEngine 获取所有对讲组的接口, 舍弃同步获取我当前所有组的接口;	刘 扬, 阳坤
1.0.1.3	1. 增加了是否需要多路呼叫, 默认需要。具体 API 请看 CallEngine; 2. 修改电话记录重复问题; 3. C++ 完善硬编码, 应用只需传入 NV21 格式的数据即可。	刘 扬, 阳坤
1.0.1.4	1. ContactsEngine 增加 getUser() 获取用户信息 API;	刘 扬, 阳坤
1.0.1.5	1. 增加 MessageEngine 消息加载更多接口, 解决加载 BUG; 2.增加获取当前与谁聊天的历史消息总数量	刘 扬, 阳坤
1.0.1.6	开放 G729 格式录音功能 API,详细 API 接口请看 MessageEngine	刘 扬, 阳坤
1.0.1.7	1.增加获取录音实时分贝, 详细 API 接口请看 MessageEngine#startRecordAudio; 2. 增加停止播放录音接口	刘 扬, 阳坤
1.0.1.8	修改 SDK 聊天异常 BUG	刘 扬, 阳坤
1.0.1.9	优化即时通讯 sendMessage 接口, 并支持重发 isReSend	刘 扬, 阳坤
1.0.2.0	增加上传文件回调	刘 扬, 阳坤
1.0.2.1	解决通话异常 BUG	刘 扬, 阳坤
1.0.2.4	1.PttEngine 增加 changePttCurState 接口 2. 解决通话状态 call_type 不更新问题 3. 增加通话详细历史字段 4. rtp 重复 10次为 1 次。	刘 扬, 阳坤
1.0.2.5	1.增加 音视频来电 声音大小控制接口, 详细请看 <code>SetEngine#onKeyDown</code> 函数; 2. 解决多路通话历史记录异常问题。	刘 扬, 阳坤
1.0.2.6	1. 增加多路音频的操作(静音, 播放功能), 详细使用请看 CallEngine	刘 扬, 阳坤
1.0.2.7	1. 将 so 放入 aar 中, 无需在外部导入。2. 增加 PTT (录音回放/PCM 播放)接口详细请看 PttEntine 、 DEMO	刘 扬, 阳坤
1.0.2.8	1. 增加会议启动成功返回了一个 会议 id 值(MeetingCallBack##onGetMettingSuccess(String id))。2. 增加了	刘

	对语音会议(邀请入会，踢人，禁言)等接口。	刘扬， 阳坤
1.0.2.9	1. 修改了PTT本地录制 BUG,2. 修改了 pushYUV 为 pushH264 接口。	刘扬， 阳坤
1.0.3.0	1. 增加 PTT 回放时间段查询 PttEngine#getGroupAndTalkToPttPlaybackDataAsyn(int groupId, int talkId, String startTime,String stopTime,PttAudioHistoryDataDao.IFindCallback iFindCallback)	刘扬， 阳坤
1.0.4.0	1. 修复隐藏 bug；2. 增加切换用户接口；3. 增加多路通话设置混音接口	刘扬， 阳坤
1.0.4.8	修复 bug	刘扬， 阳坤

USBCamera SDK

版本	说明	备注
v1.2	增加单独的 USB 摄像头 SDK	刘扬、阳坤
v1.3	解决异常 Crash	刘扬、阳坤

标准版本嘀嗒 APK 扫码下载:



注意

华为手机 APP 长连接 异常问题

//华为手机使用嘀哒软件在后台运行出现人员掉线问题解决

手机 => 电池 > 更多设置 > 休眠保持网络连接

手机 => 电池 > 启动应用 > 切换为手动管理（允许自启动，允许关联启动，允许后台活动）

快速集成

1. 将 .aar SDK 放入 app/lib ,或者 module/lib 中
2. 将 so 库放入 src/main/jniLibs 目录 并配置 ndk
SDK > 1.0.2.7 不需要外部放入 so
3. 在 module/build.gradle 或者 app/build.gradle 中配置

```
android{
    ...

    defaultConfig {
        ....

        //设置支持的 so 库支持
        ndk {
            abiFilters 'armeabi-v7a'
        }
    }
    allprojects {
        repositories {
            flatDir {
                //dirs '../t01_module/libs'; //多模块开发参考这种集成方式。
                dirs 'libs';
            }
        }
    }
}

dependencies{
    ...
    compile(name:'LY_AudioVideoHelp_SDK_debug', ext:'aar') //名字以 aar 名字为准
}
```

4. 继承 PttApplication
5. 添加 `AndroidManifest.xml` 配置

```

<service android:name="com.bnc.activity.service.MessengerService" />
<service android:name="com.bnc.activity.view.NativeService" />
<receiver android:name="com.bnc.activity.receiver.SipEventReceiver">
    <intent-filter>
        <action android:name="com.bnc.app.action.INCOMING" />
        <action android:name="com.bnc.app.action.SMS" />
    </intent-filter>
</receiver>

```

6. 初始化 SDK

```

T01Helper.getInstance().initT01SDK(Context context);

//推荐初始化
private void initT01SDK() {
    //初始化 SDK
    T01Helper.getInstance().initT01SDK(getApplicationContext());
    //需要多路 true
    T01Helper.getInstance().getCallEngine().setMultipleLines(true);
    //交给Java硬编码
    T01Helper.getInstance().getSetEngine().setJavaMediacodec(true);
    //0:320P 1:480P 2:720P 3:1080P
    T01Helper.getInstance().getSetEngine().setVideoCallInCallQuality(2);
    if (T01Helper.getInstance().getCallEngine().isMultipleLines()) {
        //设置多路全部禁言模式 (true:不推音频流->对端听不见发送端的声音, false :推送音频流)
        T01Helper.getInstance().getCallEngine().setMoreAudioMute(false);
        //设置多路全部播放模式 (true:播放->播放对端的声音, false :不播放对端声音)
        T01Helper.getInstance().getCallEngine().setMoreAudioPlay(true);
        //多路呼叫需要混音
        T01Helper.getInstance().getSetEngine().setEnableMixer(true);
    }
}

```

7. 退出 SDK

```

void exitT01SDK(ILoginOutListener loginOutListener)

```

8. 在 module/build.gradle 或者 app/build.gradle 中配置 USB Camera SDK

```

android{
    ...

    defaultConfig {
        ....

        //设置支持的 so 库支持
    }
}

```

```

        ndk {
            abiFilters 'armeabi-v7a'
        }
    }
    allprojects {
        repositories {
            flatDir {
                //dirs '../t01_module/libs'; //多模块开发参考这种集成方式。
                dirs 'libs';
            }
        }
    }
}

dependencies{
    ...
    compile(name: 'T01_AV_USBCamera_2020-03-19-12-48_v1.2_', ext: 'aar') //名字以 aar
    名字为准
}

```

9. 配置混淆规则

```

-dontwarn com.bnc.activity.**
-dontwarn org.apache.http.**
-keep class com.bnc.activity.** { *; }
-keep class org.doubango.tinyWRAP.** { *; }
-keep class org.apache.http.** { *;}

```

RegisterEngine

- 是否注册

```
boolean isRegister()
```

- 登录

```
void login(String userId, String pwd, String server_ip, String server_port, final
LoginCallBack loginCallBack)
```

PttEngine

- 获取当前对讲组列表


```
void getCurrentPttGroup(final PttListCallBack pttListListener)
```

- PttListCallBack

```
/**
 * 对讲组信息回调
 */
public interface PttListCallBack {

    /**
     * 获取当前组列表
     *
     * @param userList
     */
    void getCurrentPttLists(ArrayList<UserEntity> userList);

    /**
     * PttVOIP 登录回调 对讲状态
     */
    void voipLoginState(String msg);
}
```

- 开始对讲

```
void startPttGroup();
```

- 停止对讲

```
void stopPttGroup();
```

- 获取所有对讲组

```
getAllPttGroupLists(final IAllPttGroupCallBack pttListListener)
```

- IAllPttGroupCallBack

```
public interface IAllPttGroupCallBack{

    /**
     * 获取所有组列表
     *
     * @param list
     */
    void getAllPttLists(ArrayList<GroupEntity> list);
}
```


- 切换对讲组

```
String setCurrentPttGroup(int groupId);
```

- 获取上一次切换的对讲组

```
int getHistoryChangePttGroup();
```

- 创建临时对讲组

```
void createTempPttGroup(String tempGroupName, final ArrayList<Integer> selectList,  
final ICreateTempListener iCreateTempListener);
```

- 删除临时对讲组

```
void delTmpGroup(int groupId, TempGroupManager.IPttDeleteUpDataListener  
iPttDeleteUpDataListener);
```

- 获取对讲通讯录

```
void getContactsLists(final IContactsListener callViewBackContact);
```

- 单呼

```
void sendOnePttCall(boolean isCall, int number);
```

- 是否自动切换对讲组

```
autoChangePtt(boolean isAuto);
```

- 判断当前对讲组是否是空闲状态

```
boolean getCurPttState();
```

- 当前对讲组正在说话的用户

```
int getCurPttCall();
```

- 对正在进行对讲组说话的通道进行操作(这个 API 不是公共的)

```
//1,监听, 2 取消监听, 3 强拆, 4 强插  
changePttCurState(int type, int groupId, TempGroupManager.IPttStateListener  
iPttStateListener)
```

- 查询所有的对讲回放数据

```
//同步， 根据时间降序返回
List<PttHistoryEntity> getAllPttAudioHistory()
//异步
void getAllPttAudioHistoryAsync(PttAudioHistoryDataDao.IFindCallback iFindCallback)
```

- 删除所有数据

```
//同步， 返回共总删除的条目数量
int delAllPlaybackData();
//异步
void delAllPlaybackDataAsync(PttAudioHistoryDataDao.IDelCallback iDelCallback)
```

- 根据对应的组 ID 来查找数据

```
//同步
List<PttHistoryEntity> getPttGroupAudioHistory(int groupId)
//异步
void getPttGroupAudioHistory(int groupId, PttAudioHistoryDataDao.IFindCallback iFindGroupCallback)
```

- 根据对应的组 ID 和说话 ID 来找历史数据

```
//同步
List<PttHistoryEntity> getGroupAndTalkToPttPlaybackData(int groupId, int talkId)
//异步
void getGroupAndTalkToPttPlaybackDataAsync(int groupId, int talkId,
PttAudioHistoryDataDao.IFindCallback iFindCallback);
```

- 查询所有组对应的数据，二级组织

```
void getAllGroupPttPlaybackDataAsync(PttAudioHistoryDataDao.IFindGroupCallback iFindGroupCallback)
```

- 动态通知存储回放数据又更新了，你需要更新 UI

```
//异步
void addPttPlaybackDataListener(PttAudioHistoryDataDao.IPttPlaybackDataChange iPttPlaybackDataChange)
```

- 播放PTT音频文件

```
void playPttPlayback(String audioPath, AudioTracker.IPlayListener iPlayListener)
```

- 停止播放PTT 音频文件

```
void stopPttPlayback()
```

- 根据时间段查询 PTT 回放数据

```
/**
 *
 * @param groupId 填 0 表示 查询所有
 * @param talkId 填 0 表示 查询所有
 * @param startTime 格式 "yyyy-MM-dd HH:mm:ss"
 * @param stopTime 格式 "yyyy-MM-dd HH:mm:ss"
 * @param iFindCallback
 */
public void getGroupAndTalkToPttPlaybackDataAsync(int groupId, int talkId, String
startTime,String stopTime,PttAudioHistoryDataDao.IFindCallback iFindCallback);
```

- 操作 PTT 回放数据返回 PttHistoryEntity 实体类说明

```
/**
 * 当前讲话人所在组 ID
 */
public int groupId;
/**
 * 当前讲话人所在组 name
 */
public String groupName;
/**
 * 当前讲话人
 */
public String talkName;

/**
 * 当前讲话人 ID
 */
public int talkId;

/**
 * 当前讲话人录音文件
 */
public String recordAudioPath;
```

```

/**
 * 保存的时间
 */
public String recordAudioTime;

/**
 * 类型 组呼/单呼/全呼
 *
 *
 **/
/**
 * 单呼
 * int CALL_SINGLE_MSG = 0xd1;
 */

/**
 * 组呼
 * CALL_MULTI_MSG = 0xd2;
 */

/**
 * 全呼
 * CALL_ALL_MSG = 0xd8;
 */
public int audioType;

```

CallEngine

- 设置相机旋转的角度（一般不用设置）

```
setCamRotation(int rotation);
```

- 获取相机旋转的角度

```
int getCamRotation();
```

- 呼出/呼入 电话监听

```
addCallReceiverListener(final CallEngine.ICallEventCallBack callEventListener)
```

- CallEngine.ICallEventCallBack

```

/**
 *会议
 */

```

```

void onCallMeetingComing(CALL_TYPE var1, String var2, NgnAVSession var3);

/**
 *呼出回调
 */
void onCallOutComing(CALL_TYPE var1, String var2, NgnAVSession var3);

/**
 *来电回调
 */
void onCallInComing(CALL_TYPE var1, String var2, NgnAVSession var3);

/**
 *监控回调
 */
void onVideoMonitor(CALL_TYPE var1, String var2);

/**
 *通话或者呼叫失败回调
 */
void onCallError(CALL_TYPE var1, String var2, long sessionId);

/**
 *通话结束回调
 */
void onTerminated(CALL_TYPE var1, String var2, NgnAVSession var3);

/**
 *正在通话中回调
 */
void onCallInCall(CALL_TYPE var1, String var2, NgnAVSession var3);

```

- CALL_TYPE

```

VIDEO_CALL_IN, //视频来电
VIDEO_CALL_OUT, //视频呼出
AUDIO_CALL_IN, //语音来电
AUDIO_CALL_OUT, //语音呼出
AUDIO_MEETING, //语音会议
VIDEO_MEETING, //视频会议
VIDEO_MONITOR, //视频监控-暂时用不到
AUDIO_TERMINATED, //语音通话结束
VIDEO_TERMINATED, //视频通话结束
VIDEO_CALL_IN_CALL, //视频通话中...
AUDIO_CALL_IN_CALL, //语音通话中...
ERROR, //通话失败的回调

```

- 挂断电话

```
boolean hangUpCall();
```

- 挂断电话(多路电话使用此接口)

```
boolean hangUpCall(NgnAVSession ngnAVSession)
```

- 接听电话(多路电话使用此接口)

```
boolean acceptCall(NgnAVSession ngnAVSession);
```

- 接听电话

```
boolean acceptCall();
```

- 是否开启免提(多路电话使用)

```
void isHandsfree(NgnAVSession ngnAVSession, boolean isHandsfree)
```

- 是否开启免提

```
void isHandsfree(boolean isHandsfree) ;
```

- 获取当前免提状态

```
boolean getHandsfreeState();
```

- 是否静音(多路电话使用)

```
boolean isMute(NgnAVSession ngnAVSession, boolean mute)
```

- 是否静音

```
boolean isMute();
```

- 预览本地视频

```
/**isEx: 如果填写 true 的话需要自己调用 pushH264 API 自己传输 视频流数据*/  
void startPreviewLocalVideo(FrameLayout localVideoPreview, boolean isEx);
```

- 根据线路预览视频(多路电话使用)

```
void startPreviewRemoteVideo(NgnAVSession ngnAVSession, FrameLayout  
remoteVideoPreview)
```

- 预览对方视频

```
void startPreviewRemoteVideo(FrameLayout remoteVideoPreview) ;
```

- 停止预览

```
void stopPreviewVideo();
```

- 切换摄像头

```
boolean changeCamera();
```

- 关闭本地摄像头

```
Boolean stopLocalVideo();
```

- 打电话

```
/**launchType : 视频电话: VOIP_LAUNCH_TYPE_VIDEO, 语音电话: VOIP_LAUNCH_TYPE_TELE, 语音会议: VOIP_LAUNCH_TYPE_TELECONFERENCE, 视频会议: VOIP_LAUNCH_TYPE_VIDEOCONFERENCE */
void call(String num, int launchType, String callName)
```

- 获取通话记录

```
void getCallHistoryList(final ICallHistoryDataCallBack iCallHistoryDataCallBack);
```

- 获取视频通话中视频详细信息

```
String getVideoCallInfo();
```

- 获取对方分辨率

```
String getRemoteDisplay();
```

- push H264 视频编码格式

```
pushH264(byte[] videoFrame, int width, int height);
```

- 配置对应的 Activity 声明周期

```
@Override
protected void onStart() {
    super.onStart();
    T01Helper.getInstance().getCallEngine().onCallStart();
}
```



```

}

@Override
protected void onResume() {
    super.onResume();
    T01Helper.getInstance().getCallEngine().onResume();
}

@Override
protected void onPause() {
    T01Helper.getInstance().getCallEngine().onCallPause();
    super.onPause();
}

@Override
protected void onStop() {
    T01Helper.getInstance().getCallEngine().onCallStop();
    getWindow().clearFlags(
        WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    super.onStop();
}

```

- 根据线路 ID 获取线路实体类

```

NgnAVSession getNgnAVSession(long sessionId);

```

- 释放线路

```

releaseSession(NgnAVSession ngnAVSession)

```

- 判断线路是否存在，线路是打电话的时候实例化的

```

boolean isSessionAlive(long id)

```

- 获取当前线路 Size

```

int getSessionSize();

```

- set 是否支持多路

```

setMultipleLines(boolean MultipleLines)

```

- 多路音频(禁言，播放操作)[详细代码请看源码 DEMO](#)

```

//API 调用
//1. Application 中初始化

```

```

if (T01Helper.getInstance().getCallEngine().isMultipleLines()) {
//设置多路全部禁言模式 (true:不推音频流->对端听不见发送端的声音, false :推送音频流;
T01Helper.getInstance().getCallEngine().setMoreAudioMute(true);
//设置多路全部播放模式 (true:播放->播放对端的声音, false :不播放对端声音)
T01Helper.getInstance().getCallEngine().setMoreAudioPlay(false);
}

//2. 对线路使用禁言功能
NgnProxyMoreAudioProducer findAudioPushLine(int sessionID);
//true 使用禁言 false 不禁言
NgnProxyMoreAudioProducer#setOnMute(boolean mute);
//当前线路是否禁言
NgnProxyMoreAudioProducer#boolean isOnMute()

//3. 对线路使用播放功能
NgnProxyAudioConsumer findAudioPlayLine(int sessionID);
//play: true: 播放, false 不播放
NgnProxyAudioConsumer#setAudioPlay(boolean play);
//当前线路是否播放
NgnProxyAudioConsumer#boolean getAudioState();

```

MeetingEngine

- 获取会议列表

```

void getMeetingList(final MeetingCallBack meetingListsListener)c

```

- 创建会议组

```

//meetingtype 0 语音会议, 1 视频会议
void launchMeeting(int meetingtype, String groupName, ArrayList<Integer>
meetingMember, MeetingCallBack meetingListsListener);

```

- 对会议中的人员操作禁言

```

/**
 *
 * @param isMute 是否禁言
 * @param conferenceId
 * @param muteType 禁言类型:
 *             = MeetingHandle.ALL 禁言/恢复通话全部参与人员
 *             = MeetingHandle.NON_MODERATOR 禁言/恢复通话除会议主持人的其他人
 *             = userID 禁言某个人ID
 * @param handleListener
 */
void meetingMemberMuteHandle(boolean isMute, String conferenceId, int muteType,
MeetingManager.IMeetingMemberMuteListener handleListener)

```

- 踢人

```

/**
 *
 * @param conferenceId
 * @param delType = MeetingHandle.ALL 踢出全部参与人员
 *               = MeetingHandle.NON_MODERATOR 踢出除会议主持人的其他人
 *               = userID 踢出某个人
 * @param handleListener
 */
void delMeetingMember(String conferenceId, int delType,
MeetingManager.IMeetingDelListener handleListener)

```

- 邀请入会

```

/**
 * 加入会议
 * @param joinMember 加入会议人的 ID
 * @param conferenceId 会议 ID
 * @param iMeetingHandleListener
 */
void joinMeeting(int joinMember, String conferenceId,
MeetingManager.IMeetingJoinListener iMeetingHandleListener)

```

- 会议组成员状态监听

```

void addMeetingStateListener(String conferenceId,
MeetingManager.IMeetingStateListener iMeetingHandleListener)

```

- 停止会议

```

void stopMeeting(String conferenceId, MeetingManager.IMeetingStopListener
iMeetingHandleListener)

```

MessageEngine

- 发送消息

```
//messageContentType MsgUtil.ImgType.xx    0 :txt,1:视频,3:图片,4:录音,10:文件
//chatType_1_or_2 1:单聊 , 2 群聊
//content: 发送的内容
//receiverId: 接收消息者的 ID
//receiverName: 接收消息者的姓名
//filePath : 文件路径
//sendVoicelong: 录音时长
//uuid : 消息唯一值,可填,可不填-----如果消息重发需要带上重发消息的 uuid
//isReSend: 是否重发;
MsgMessageEntity sendMessage(int messageContentType, String content, int
receiverId, String receiverName, String filePath, int chatType_1_or_2, String
sendVoicelong, String uuid,boolean isReSend)
```

- 删除消息

```
boolean deleteMessage(String uuid)
```

- 获取发送消息的监听

```
//sendType: 1:单聊 , 2 群聊
//targetUserId : 加载与谁的聊天记录
void recvMessageListener(final IRecvMessageListener iRecvMessageListener, int
sendType, final int targetUserId);

public interface IRecvMessageListener {
    //uniqueID: 消息唯一值
    //消息发送失败
    void setSEND_MSG_ERROR(String uniqueID, int reason);
    //消息发送成功
    void setSEND_MSG_SUCCEED(String uniqueID);
    //接收当前与 targetUserId 的消息
    void getCurrentRevMeg(MsgMessageEntity messageEntity);

    void getUserIsOnline(boolean targetUserIsOnline, boolean curUserIsOnline);
    //第一次进来默认加载与 targetUserId 的所有消息
    void getAllCurrentMeg(ArrayList<MsgMessageEntity> datas,int count);
    //加载更多消息
    void getMoreMeg(ArrayList<MsgMessageEntity> more, int size);
    //上传文件的进度 messageContentType : {@link MsgUtil.ImgType} 对比
    void postFileProgress(double pro, String uuid, int messageContentType);
}
```

- 离开聊天室

```
void onChatUiStateOnDestory();
```

- 聊天室不可见的情况

```
void onChatUiStateOnPause(Class c, String targetUserName, String sendType)
```

- 加载默认消息

```
//sendType: 1:单聊 , 2 群聊  
//nowDataCount : 默认加载消息数量  
//targetUserId : 加载与谁的聊天记录  
void loadDefaultMeg(int sendType, int nowDataCount, int targetUserId);
```

- 更新附件地址

```
void sendUpdateAttachPathRequest(MsgMessageEntity entity,  
                                boolean isDownLoad);
```

- 获取 IM 聊天会话列表

```
void getIMLists(final IMListsCallBack imListsCallBack);
```

- 后台消息监听

```
void showMegToNotity(final IShowNotityCallBack iShowNotity);
```

- 加载更多聊天消息

```
//sendType: 1:单聊 , 2 群聊  
//nowDataCount : 每次加载历史消息数量  
//targetUserId : 加载与谁的聊天记录  
//startIndex : 从数据库的那个位置开始加载历史消息  
void loadMoreMeg(int sendType, int startIndex, int nowDataCount, int targetUserId)
```

- 根据 target 获取与他聊天历史消息总数量

```
//targetUserId : 与谁的聊天记录  
//sendType: 1:单聊 , 2 群聊  
int getTargetMessageCount(int targetUserId, int sendType)
```

- 录制 G729 格式音频

```
boolean startRecordAudio(IMsgRecordListenter recordListenter)
```

```

T01Helper.getInstance().getMessageEngine().startRecordAudio(new
MsgRecordManager.IMsgRecordListener() {

    //file: 录音得到的文件
    //recordTime : 录音时长
    @Override
    public void onMessageRecordEnd(File file, long recordTime) {

        }

    @Override
    public void onVolume(String volume) {
        Log.d(TAG, "当前声音分贝: "+volume);
    }
});

```

- 停止录制

```
public void stopRecordAudio()
```

- 取消录制

```
void onCancelRecord()
```

- 播放 G729 格式音频

```
boolean playRecordAudio(String path)
```

- 停止播放 G729

```
void stopPlayRecordAudio();
```

ContactsEngine

- 查询当前组织ID

```
String getCurrentUnitId()
```

- 获取所有的组织

```

/**不传 (null or "") 默认获取所有的组织*/
void getALLUnitList(String unitID,IUnitListener iUnitListener);

```

- 获取 GPS 数据

```
void getGPSInfoList(IGPSListener igpsInfoCallBack);
```

- 获取所有用户

```
void getUserList(IUserListener iUserInfoCallBack);
```

- 获取所有外部用户

```
void getWBUserList(IWBUserListener iWbInfoCallBack);
```

- 获取在线用户

参数一：repeatTime 轮训请求以毫秒为单位, 不需要轮训请求传入 0 就行

```
void getOnlineUserCallBack(int repeatTime, IOnlineUserListener iPttDataListener)
```

- 根据 ID 获取用户信息

```
UserEntity getUser(String userId)
```

- 获取当前用户 ID

```
String getCurrentUserVoipId()
```

LocationEngine

- 发送定位数据

```
void sendLocation(int m_userLoginId,double longitude, double latitude,double  
derect,double speed,int location_type);
```

SetEngine

- 对讲音量增强

```
void setVoiceZoom(int i); //1,4,8,12,16
```

- 获取当前对讲音量

```
int getVoiceZoom(Context mContext);
```


- 获取当前视频编码

```
String getCurVideoCoding();
```

- 获取当前语音编码

```
String getCurVoiceCoding();
```

- 获取当前视频质量

```
String getCurVideoQuality();
```

- 获取当前视频监控质量

```
String getCurVideoJKQuality();
```

- 获取当前摄像头是否是前置

```
boolean isCameraFront();
```

- 设置摄像头前置或者后置

```
void setCameraFrontOrAfter(boolean isFront) ;
```

- 设置视频参数

```
//0, h264 , 1,h263 ,2 MPEG4  
void setVideoParameter(int type);
```

- 设置通话质量

```
// 0：一般，1：标清，2：高清，3：超清，4：2K  
void setVideoCallInCallQuality(int type);
```

- 设置视频监控质量

```
void setVideoJKQuality(int type);
```

- 修改密码

```
void doUpdatePassWord(String oldPassWord, String updatePassWord, final  
IUpdatePwdListener iUpdatePwdListener);
```

- 通话音量输入增益

```
void setMediaInGain(int v); // 0 - 5;
```

- 通话音量输出增益

```
void setMediaOutGain(int v) ;
```

- 获取通话音量输入增益

```
int getMediaInGain();
```

- 获取通话音量输出增益

```
int getMediaOutGain();
```

- 音视频来电声音控制

```
//1. 重写 Activity onKeyDown 方法
//2. 返回 T01Helper.getInstance().getSetEngine().onKeyDown(keyCode,event);
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    return T01Helper.getInstance().getSetEngine().onKeyDown(keyCode,event);
}
```

- 多路通话是否混音

```
void setEnableMixer(boolean ismix)
```

USBCamera SDK 使用

1. 当前使用 USB Activity layout 中添加如下布局代码

```
<include layout="@layout/layout_usbcamera"></include>
```

2. 使用

```
//1. 设置显示 USB Camera View
decorView.findViewById(R.id.fl_usb_camera).setVisibility(VISIBLE);

//2. 设置预览框口大小, 不设置默认全屏
UVCCameraTextureView uvccameraTextureView =
decorView.findViewById(R.id.camera_view);
FrameLayout.LayoutParams layoutParams = (FrameLayout.LayoutParams)
uvccameraTextureView.getLayoutParams();
layoutParams.gravity = Gravity.RIGHT;
```

```
        layoutParams.width = 500;
        layoutParams.height = 500;
        uvcCameraTextureView.setLayoutParams(layoutParams);

//3. 添加连接 USB Camera 服务监听
USBCameraHelper.getInstance(this).addUSBCameraListener(IUSBCameraConnectListener
listener);

//4. 初始化 decorView 当前 Activity 根布局
USBCameraHelper.getInstance(this).init(decorView, 1280, 720);

//5. 开始预览
USBCameraHelper.getInstance(this).onStart();

//6. 预览 NV21 数据回调
USBCameraHelper.getInstance(this).setOnPreviewFrameListener(OnPreViewResultListener
listener);
```

3. 销毁

```
USBCameraHelper.getInstance(this).onDestroy();
```